Homework 2: Sentiment

Course: CS 221 Spring 2019
Name: Bryan Yaggi

Problem 1: Building Intuition

Rotten Tomatoes has classified these reviews as "positive" and "negative", respectively, as indicated by the intact tomato on the left and the splattered tomato on the right. In this assignment, you will create a simple text classification system that can perform this task automatically. We'll warm up with the following set of four mini-reviews, each either labeled positive (+1) or negative (−1):

1. (−1) pretty bad
2. (+1) good plot
3. (−1) not good
4. (+1) pretty scenery

Each review x is mapped onto a feature vector $\phi(x)$, which maps each word to the number of occurrences of that word in the review. For example, the first review maps to the (sparse) feature vector $\phi(x) = \{pretty : 1, bad : 1\}$. Recall the definition of the hinge loss:

$$Loss_{hinge}(x, y, \mathbf{w}) = max\{0, 1 - \mathbf{w} \cdot \phi(x)y\},$$

where y is the correct label.

(a) Suppose we run stochastic gradient descent, updating the weights according to

$$w \leftarrow w - \eta \nabla_w Loss_{hinge}(x, y, \mathbf{w}),$$

once for each of the four examples in the order given above. After the classifier is trained on the given four data points, what are the weights of the six words ("pretty", "good", "bad", "plot", "not", "scenery") that appear in the above reviews? Use $\eta = .5$ as the step size and initialize $w = [0, \ldots, 0]$. Assume that $\nabla_w Loss_{hinge}(x, y, \mathbf{w}) = 0$ when the margin is exactly 1. “”

Let the feature and weights vector be the number of times “pretty”, “bad”, “bad”, “plot”, “not”, and “scenery” occur in x, respectively.

$$x_1 = \text{“}pretty\ bad\text{”}, \phi(x_1) = \{\text{"}pretty\text{"} = 1, \text{"}bad\text{"} = 1\}, y_1 = -1$$
$$x_2 = \text{“}good\ plot\text{”}, \phi(x_2) = \{\text{"}good\text{"} = 1, \text{"}plot\text{"} = 1\}, y_2 = +1$$
$$x_3 = \text{“}not\ good\text{”}, \phi(x_3) = \{\text{"}not\text{"} = 1, \text{"}good\text{"} = 1\}, y_3 = -1$$
$$x_4 = \text{“}pretty\ scenery\text{”}, \phi(x_4) = \{\text{"}pretty\text{"} = 1, \text{"}scenery\text{"} = 1\}, y_4 = +1$$

$$\frac{\partial Loss_{hinge}}{\partial w} = \begin{cases} -\phi(x)y, & w \cdot \phi(x)y < 1 \\ 0, & w \cdot \phi(x)y \geq 1 \end{cases}$$

$$w_{after\ step\ 1} = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix} - .5\begin{bmatrix}(-1)(-1)\\0\\(-1)(-1)\\0\\0\\0\end{bmatrix} = \begin{bmatrix}-.5\\0\\-.5\\0\\0\\0\end{bmatrix}$$

$$w_{after\ step\ 2} = \begin{bmatrix}-.5\\0\\-.5\\0\\0\\0\end{bmatrix} - .5\begin{bmatrix}0\\(-1)(1)\\0\\(-1)(1)\\0\\0\end{bmatrix} = \begin{bmatrix}-.5\\.5\\-.5\\.5\\0\\0\end{bmatrix}$$

$$w_{after\ step\ 3} = \begin{bmatrix}-.5\\.5\\-.5\\.5\\0\\0\end{bmatrix} - .5\begin{bmatrix}0\\(-1)(-1)\\0\\0\\(-1)(-1)\\0\end{bmatrix} = \begin{bmatrix}-.5\\0\\-.5\\.5\\-.5\\0\end{bmatrix}$$

$$w_{after\ step\ 4} = \begin{bmatrix}-.5\\0\\-.5\\.5\\-.5\\0\end{bmatrix} - .5\begin{bmatrix}(-1)(1)\\0\\0\\0\\0\\(-1)(1)\end{bmatrix} = \begin{bmatrix}0\\0\\-.5\\.5\\-.5\\.5\end{bmatrix}$$

(b) Create a small labeled dataset of four mini-reviews using the words "not", "good", and "bad", where the labels make intuitive sense. Each review should contain one or two words, and no repeated words. Prove that no linear classifier using word features can get zero error on your dataset. Remember that this is a question about classifiers, not optimization algorithms; your proof should be true for any linear classifier, regardless of how the weights are learned. After providing such a dataset, propose a single additional feature that we could augment the feature vector with that would fix this problem. (Hint: think about the linear effect that each feature has on the classification score.)

Let ϕ_1, ϕ_2, ϕ_3 be the number of times "bad", "good", and "not" occur in x, respectively.

$$\begin{aligned}
x_1 &= \text{“}bad\text{”}, \phi_1 = \{\text{”}bad\text{”} = 1\}, y_1 = -1\\
x_2 &= \text{“}good\text{”}, \phi_2 = \{\text{”}good\text{”} = 1\}, y_2 = +1\\
x_3 &= \text{“}not\ bad\text{”}, \phi_3 = \{\text{”}not\text{”} = 1, \text{”}bad\text{”} = 1\}, y_3 = +1\\
x_4 &= \text{“}not\ good\text{”}, \phi_4 = \{\text{”}not\text{”} = 1, \text{”}good\text{”} = 1\}, y_4 = -1
\end{aligned}$$

$$\begin{bmatrix}1&0&0\\0&1&0\\1&0&1\\0&1&1\end{bmatrix} \cdot \begin{bmatrix}w_1\\w_2\\w_3\end{bmatrix} - \begin{bmatrix}-1\\1\\1\\-1\end{bmatrix} \neq 0$$

Add a feature for word count. The feature will be 0 if x has one word, and 1 if x has two words.

$$x_1 = \text{“}bad\text{”}, \phi_1 = \{\text{”}bad\text{”} = 1\}, y_1 = -1$$
$$x_2 = \text{“}good\text{”}, \phi_2 = \{\text{”}good\text{”} = 1\}, y_2 = +1$$
$$x_3 = \text{“}not\ bad\text{”}, \phi_3 = \{\text{”}not\text{”} = 1, \text{”}bad\text{”} = 1, count = 1\}, y_3 = +1$$
$$x_4 = \text{“}not\ good\text{”}, \phi_4 = \{\text{”}not\text{”} = 1, \text{”}good\text{”} = 1, count = 1\}, y_4 = -1$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = 0$$
$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \ or \ \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

Problem 2: Predicting Movie Ratings

Suppose that we are now interested in predicting a numeric rating for each movie review. We will use a non-linear predictor that takes a movie review x and returns $\sigma(\mathbf{w} \cdot \phi(x))$, where $\sigma(z) = (1 + e^{-z})^{-1}$ is the logistic function that squashes a real number to the range $(0, 1)$. Suppose that we wish to use the squared loss. For this problem, assume that the movie rating y is a real-valued variable in the range $[0, 1]$.

(a) Write out the expression for $Loss(x, y, \mathbf{w})$.

$$\begin{aligned} Loss_{squared}(x, y, \mathbf{w}) &= (f_w(x) - y)^2 \\ &= (\sigma(\mathbf{w} \cdot \phi(x)) - y)^2 \\ &= ((1 + e^{-w \cdot \phi(x)})^{-1} - y)^2 \end{aligned}$$

(b) Compute the gradient of the loss with respect to w. Hint: you can write the answer in terms of the predicted value $p = \sigma(\mathbf{w} \cdot \phi(x))$.

$$\begin{aligned} \nabla_w Loss_{squared}(x, y, \mathbf{w}) &= 2(p - y)\frac{\partial p}{\partial \mathbf{w}} \\ &= 2((1 + e^{-w \cdot \phi(x)})^{-1} - y)(-(1 + e^{-w \cdot \phi(x)})^{-2})(e^{-w \cdot \phi(x)})(-\phi(x)) \\ &= 2((1 + e^{-w \cdot \phi(x)})^{-1} - y)(1 + e^{-w \cdot \phi(x)})^{-2} e^{-w \cdot \phi(x)} \phi(x) \end{aligned}$$

(c) Suppose there is one datapoint (x, y) with some given $\phi(x)$ and $y = 1$. Can you choose a $\mathbf{w}$ to make the magnitude of the gradient of the loss with respect to $\mathbf{w}$ arbitrarily small (i.e., minimize the magnitude of the gradient and make it asymptotically approach some value)? If so, how small? Can the magnitude of the gradient ever be exactly zero? You are allowed to make the magnitude of $\mathbf{w}$ arbitrarily large. Hint: try to understand intuitively what is going on and the contribution of each part of the expression. If you find yourself doing too much algebra, you're probably doing something suboptimal. Motivation: the reason why we're interested in the magnitude of the gradients is because it governs how far gradient descent will step. For example, if the gradient is close to zero when $\mathbf{w}$ is very far from the optimum, then it could take a long time for gradient descent to reach the optimum (if at all). This is known as the vanishing gradient problem when training neural networks.

$$\begin{aligned}\lim_{w\to\infty} \nabla_w Loss_{squared}(x, y, \mathbf{w}) &= 0\\ \text{since} \lim_{w\to\infty} e^{-w\cdot\phi(x)} &= 0\\ \lim_{w\to-\infty} \nabla_w Loss_{squared}(x, y, \mathbf{w}) &= 0\\ \text{since} \lim_{w\to-\infty} ((1 + e^{-w\cdot\phi(x)})^{-2} &= 0\end{aligned}$$

The gradient will asymptotically reach 0, but never truly be 0.

(d) Assuming the same data point as above, what is the largest magnitude that the gradient can take? Leave your answer in terms of $\|\phi(x)\|$.

$$\begin{aligned}\|\nabla_w Loss_{squared}(x, y, \mathbf{w})\| &= 2(p - y)p^2(1 - p)\|\phi(x)\|\\ \|\nabla_w Loss_{squared}(x, y = 1, \mathbf{w})\| &= 2(p - 1)p^2(1 - p)\|\phi(x)\|\\ &= (-2p^4 + 4p^3 - 2p^2)\|\phi(x)\|\\ \nabla^2_w L_{squared}(x, y = 1, \mathbf{w}) &= -8p^3 + 12p^2 - 4p = 0 \implies p = 1, \frac{1}{2}\\ max(\|\nabla_w Loss_{squared}(x, y = 1, \mathbf{w})\|) &= -\frac{1}{8}\|\phi(x)\|\end{aligned}$$

(e) The problem with the loss function we have defined so far is that is it is non-convex, which means that gradient descent is not guaranteed to find the global minimum, and in general these types of problems can be difficult to solve. So let us try to reformulate the problem as plain old linear regression. Suppose you have a dataset $\mathbf{D}$ consisting of (x, y) pairs, and that there exists a weight vector $\mathbf{w}$ that yields zero loss on this dataset. Show that there is an easy transformation to a modified dataset $\mathbf{D}'$ of (x, y') pairs such that performing least squares regression (using a linear predictor and the squared loss) on $\mathbf{D}'$ converges to a vector $\mathbf{w}^*$ that yields zero loss on $\mathbf{D}'$. Concretely, write an expression for y' in terms of y and justify this choice. This expression should not be a function of $\mathbf{w}$.

The logit function is the inverse of the sigmoid function and converts from $[0, 1]$ to $[-\infty, +\infty]$.

$$y' = logit(y) = log(\frac{y}{1 - y})$$

Problem 3: Sentiment Classification

In this problem, we will build a binary linear classifier that reads movie reviews and guesses whether they are "positive" or "negative." In this problem, you must implement the functions without using libraries like Scikit-learn.

(a) coding

(b) coding

(c) coding

(d) When you run the grader.py on test case 3b-2, it should output a weights file and a error-analysis file. Look through some example incorrect predictions and for five of them, give a one-sentence explanation of why the classification was incorrect. What information would the classifier need to get these correct? In some sense, there's not one correct answer, so don't overthink this problem. The main point is to convey intuition about the problem.

Review 1: home alone goes hollywood , a funny premise until the kids start pulling off stunts not even steven spielberg would know how to do . besides , real movie producers aren't this nice .
Truth: -1, Prediction: 1 [WRONG]

It is tough to determine the sentiment of this review via individual words alone.

Review 2: a perfectly competent and often imaginative film that lacks what little lilo & stitch had in spades – charisma .
Truth: 1, Prediction: -1 [WRONG]

This review had several stop-words that were weighted heavily negative.

Review 3: a heady , biting , be-bop ride through nighttime manhattan , a loquacious videologue of the modern male and the lengths to which he'll go to weave a protective cocoon around his own ego .
Truth: 1, Prediction: -1 [WRONG]

The vocabulary in this review is advanced, so many of the important words did not have significant weights assigned to them. Also, there were many stop-words heavily weighted negative.

Review 4: 'it's painful to watch witherspoon's talents wasting away inside unnecessary films like legally blonde and sweet home abomination , i mean , alabama . '
Truth: -1, Prediction: 1 [WRONG]

Several of the movie title words were heavily weighted positive. "Painful" was curiously weighted heavily positive.

Review 5: dull , if not devoid of wit , this shaggy dog longs to frisk through the back alleys of history , but scarcely manages more than a modest , snoozy charm .
Truth: -1, Prediction: 1 [WRONG]

Consideration of word proximity is important in this review. Phrases like "devoid of wit", "scarcely manages", and "snoozy charm" should all be negative, but the end words are weighted heavily positive, which leads to the review's positive sentiment score.

(e) coding

(f) Run your linear predictor with feature extractor `extractCharacterFeatures`. Experiment with different values of n to see which one produces the smallest test error. You should observe that this error is nearly as small as that produced by word features. How do you explain this? Construct a review (one sentence max) in which character n-grams probably outperform word features, and briefly explain why this is so.

The predictor worked best with n between 5-7. This value is large enough to capture word proximity and most short words. If n is smaller, the n-grams are meaningless and if they are larger, they would be too unique.

This approach would work well on a review like "It's not good." The n-gram "notgood" would be weighted negative.

Problem 4: K-Means Clustering

Suppose we have a feature extractor ϕ that produces 2-dimensional feature vectors, and a toy dataset $D_{train} = \{x1, x2, x3, x4\}$ with

$$\phi(x_1) = [1,0]$$
$$\phi(x_2) = [1,2]$$
$$\phi(x_3) = [3,0]$$
$$\phi(x_4) = [2,2]$$

(a) Run 2-means on this dataset until convergence. Please show your work. What are the final cluster assignments z and cluster centers μ? Run this algorithm twice with the following initial centers:

(1) $\mu_1 = [2,3], \mu_2 = [2,-1]$
Epoch 1:
Step 1: Assign feature vectors to clusters.

$$\begin{aligned}
&\|\phi(x_1) - \mu_1\|^2 = 10\\
&\|\phi(x_1) - \mu_2\|^2 = 2 \implies z_1 = \mu_2\\
&\|\phi(x_2) - \mu_1\|^2 = 2\\
&\|\phi(x_2) - \mu_2\|^2 = 10 \implies z_2 = \mu_1\\
&\|\phi(x_3) - \mu_1\|^2 = 10\\
&\|\phi(x_3) - \mu_2\|^2 = 2 \implies z_3 = \mu_2\\
&\|\phi(x_4) - \mu_1\|^2 = 1\\
&\|\phi(x_4) - \mu_2\|^2 = 9 \implies z_4 = \mu_1
\end{aligned}$$

Step 2: Find new means for each cluster

$$\mu_1 = \frac{[1,2]+[2,2]}{2} = [\frac{3}{2}, 2]$$
$$\mu_2 = \frac{[1,0]+[3,0]}{2} = [2,0]$$

Epoch 2:
In step 1, the assignments do not change, so convergence has been reached.

(2) $\mu_1 = [0,1], \mu_2 = [3,2]$
Epoch 1:
Step 1: Assign feature vectors to clusters.

$$\begin{aligned}
&\|\phi(x_1) - \mu_1\|^2 = 2\\
&\|\phi(x_1) - \mu_2\|^2 = 8 \implies z_1 = \mu_1\\
&\|\phi(x_2) - \mu_1\|^2 = 2\\
&\|\phi(x_2) - \mu_2\|^2 = 4 \implies z_2 = \mu_1\\
&\|\phi(x_3) - \mu_1\|^2 = 10\\
&\|\phi(x_3) - \mu_2\|^2 = 4 \implies z_3 = \mu_2\\
&\|\phi(x_4) - \mu_1\|^2 = 5\\
&\|\phi(x_4) - \mu_2\|^2 = 1 \implies z_4 = \mu_2
\end{aligned}$$

Step 2: Find new means for each cluster

$$\mu_1 = \frac{[1,0]+[1,2]}{2} = [1,1]$$
$$\mu_2 = \frac{[3,0]+[2,2]}{2} = [\frac{5}{2},1]$$

Epoch 2:
In step 1, the assignments do not change, so convergence has been reached.

(b) coding

(c) Sometimes, we have prior knowledge about which points should belong in the same cluster. Suppose we are given a set S of example pairs (i, j) which must be assigned to the same cluster. For example, suppose we have 5 examples; then $S = \{(1,5),(2,3),(3,4)\}$ says that examples 2, 3, and 4 must be in the same cluster and that examples 1 and 5 must be in the same cluster. Provide the modified k-means algorithm that performs alternating minimization on the reconstruction loss:

$$\sum_i^n = \|\mu_{z_i} - \phi(x_i)\|^2$$

where μ_{z_i} is the assigned centroid for the feature vector $\phi(x_i)$. Recall that alternating minimization is when we are optimizing two variables jointly by alternating which variable we keep constant.

Before starting the k-means algorithm, a new example can be created from the mean of the examples that are known to belong to the same cluster. The original examples that are represented by new examples can be removed from the algorithm, but each of the new examples needs to be weighted by the number of original examples it represents when computing new means for each cluster. The total loss at the end will need to be calculated using only the original examples.

(d) What is the advantage of running k-means multiple times on the same dataset with the same K, but different random initializations?

Prevents the chance of only finding a local minimum.

(e) If we scale all dimensions in our initial centroids and data points by some factor, are we guaranteed to retrieve the same clusters after running k-means (i.e. will the same data points belong to the same cluster before and after scaling)? What if we scale only certain dimensions? If your answer is yes, provide a short explanation. If it is no, provide a counterexample.

Yes, if all dimensions are scaled by the same factor, the clusters will remain the same. If only certain dimensions are scaled, the clusters will not necessarily remain the same. The distances between points are dependent on all dimensions.